VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÝ KALENDÁŘ UMOŽŇUJÍCÍ EXTRÉMNĚ SNADNÉ SDÍLENÍ

WEB CALENDAR FOR EXTREMELY SIMPLE SHARING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ARINA SHARLAIEVA**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Studentka: **Sharlaieva Arina**
Program: Informační technologie
Název: **Webový kalendář umožňující extrémně snadné sdílení**
Web Calendar for Extremely Simple Sharing
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou vývoje webových aplikací s komunikací v reálném čase.
2. Vyhledejte, prostudujte a popište existující sdílené kalendáře a obdobné služby.
3. Navrhněte kalendář s extrémně jednoduchým sdílením. Identifikujte nejdůležitější funkcionalitu.
4. Prototypujte navržený kalendář a intenzivně testujte s uživateli.
5. Iterativně vylepšujte navržené a vytvořené řešení, neustále testujte.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Jan Řezáč: Web ostrý jako břitva, Baroque Partners, 2014

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Práce se zabývá návrhem a implementací webového kalendáře umožnujícího extrémně snadné sdíleni v jazyce Python s využitím frameworku Flask pro serverovou část a JavaScript a jeho knihovně React pro klientskou.

Abstract

The aim of this work was to create a web calendar application. The application was written in Python and its framework Flask on the server side and in Javascript using its library React on the client

Klíčová slova

webový kalendář, Flask, Python, JavaScript, React.

Keywords

calendar application, Flask, Python, JavaScript, React.

Citace

SHARLAIEVA, Arina. *Webový kalendář umožňující extrémně snadné sdílení.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Adam Herout, Ph.D.

Webový kalendář umožňující extrémně snadné sdílení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Arina Sharlaieva
10. května 2021

Poděkování

Ráda bych zde poděkovala prof. RNDr. Adamovi Heroutovi, Ph.D. za trpělivost a odborné vedení této práce.

Obsah

Kapitola 1

Úvod

V dnešní době se práce i komunikace více a více přesouvají na internet. Objevují se pořád nové webové aplikace, které nahrazují mnoho obvyklých akci. Pro široké publikum je teď mnohem pohodlnější používat právě webové aplikace a ukládat svá data do cloudu, protože výhody tohoto typu aplikací byly již v poslední době zřejmé. Mezi tyto výhody patří za prve možnost sdílení práce - mnoho lidí můžou pracovat na jednom dokumentu současně, dále možnost ukládat svá data na vzdáleném serveru umožňuje uživateli využívat větší objemy úložiště a také chránit vlastní data před ztrátou. Zároveň také dovoluje zaručit i bezpečnost dát díky moderním technologiím šifrování, které dnes podporuje většina webových prohlížečů.

Zde se dostáváme k další výhodě používání webových aplikací - vzhledem k tomu, že všechny tyto aplikace fungují v prohlížečích, dává to určitou nezávislost na platformě, která nám umožní pracovat s aplikací téměř z jakéhokoliv zařízení. Při pohledu na všechny tyto pozitivní aspekty je snadné pochopit, proč si webové aplikace v moderním světě získávají takovou popularitu - nejenže uspokojují potřeby moderní společnosti, ale v mnoha ohledech je formují. A protože v moderní době všechno usiluje o zjednodušení a zrychlení stejným směrem jdou i webové aplikace.

Další důležitá věc, kterou je třeba zmínit je to že aplikace nyní začínají být více a více vysoce specializované. Uživatelé stále častěji nechtějí pracovat se složitou a těžkou aplikací, jejíž zvládnutí zabere hodně času. Navíc takové aplikace nemohou potěšit všechny uživatele, a proto často obsahují zbytečné funkce. Uživatelé stále více preferují výběr odlehčených jednoduchých aplikací pro jednu, nebo několik málo, specifických funkcí, ale zároveň musí mít taková aplikace jednoduché rozhraní, nepřetížené zbytečnými funkcemi, které opět ucpou pracovní plochu. Pro mnoho lidi je pohodlnější zvolit si a přizpůsobit jednu nebo dvě aplikace pro sebe, pokud ovšem nemluvíme o profesionálních programech. Uživatel chce trávit stále méně času učením se rozhraní nové aplikace. Aplikace se optimalizují a jsou stále rychlejší. Uživatelé jsou ochotni trávit stále méně času čekáním na načtení stránky nebo odezvu některých funkcí. Stejně tak jsou uživatelé ochotni věnovat registraci stále méně času. Ten čas, který uživatel stráví od načtení stránky do zahájení práce s jejími funkcemi, by měl být minimální. Takové řešení bylo zapotřebí i mezi webovými kalendáři. Aby si každý mohl vytvořit a upravit kalendář na jedno kliknutí, a to i bez nutnosti procházet dlouhým procesem registrace.

Kapitola 2

Design webových aplikací

Po vynálezu internetu v roce 1946 design na „webových stránkách“ zcela chyběl. Byly to jen černé obrazovky s pixely a jediným dostupným designovým řešením bylo rozdělení informací do tabulek.

Webové stránky, na které jsme zvyklí, se objevily na počátku 90. let. První takovou stránku v roce 1991 vytvořil zaměstnanec Evropské organizace pro jaderný výzkum (CERN) Timothy Berners-Lee. Na této stránce popsal svůj navrhovaný koncept *World Wide Web (WWW)*, který je dnes široce známý. Tento koncept spočívá v publikaci hypertextových dokumentů propojených hypertextovými odkazy, které by usnadnily vyhledávání a organizaci informací. Design těchto stránek však stále spočíval pouze v organizování čistě textových informací ve formě tabulek a jejich odsazovaní[2]. Na stránkách stále zcela chyběly vizuální prvky. Ale již v roce 1992 vydává Timothy Berners-Lee aktualizaci prohlížeče, která podporuje fotografii. Tak se objevil první obrazek na internetu. V blízké budoucnosti se na internetu začali objevovat první inzerenti. To dalo hlavní impuls vývoji webového designu. Postupem času se reklamy na stránkách staly stále více a stránky byly silně přetíženy. Uživatelé v té době byli zvyklí na to, že když otevřou web na internetu, přivítá je obrovské množství barevné reklamy. Zlomem bylo vytvoření vyhledávače Google. Jeho tvůrci se naopak rozhodli dodržovat minimalistický design. .Hlavní stránka vyhledávače Google zůstala od svého založení v zásadě beze změny.

Postupně se tedy design začal měnit v opačném směru, od přetížení k jednoduchosti. To lze snadno vidět i na změnách v designu firemních log.

Nárůst počtu uživatelů vedl k rozvoji podnikání na internetu. Web společnosti se nyní stal její vizitkou. Rovněž vzrostla poptávka na tvorbu atraktivních ale za prvé generujících prodeje webových stránek. Stránka by nyní měla být jednoduchá, s intuitivním rozhraním a měla by uživatele povzbudit k akci nebo nákupu maximálně během několika sekund. Složitost, přetíženost nebo příliš dlouhá odezva webové stránky mohou vést k tomu, že ji uživatel jednoduše opustí.

2.1 Přehled současného stavu a trendy

Umělá inteligence

Chytrá domácnost, hlasové vyhledávání, roboti na úklid domu, autopilot a další podobné technologie jsou umělá inteligence. Každý z nich již má na trhu své vlastní ztělesnění. UI usnadňuje jakoukoli akci mnohokrát. Nejprve bude široce používán při vývoji webových služeb. Programátoři vědí, že většinu akcí lze bezpečně přenést na robota bez ztráty kva-

lity. ztracený čas se však výrazně zkrátí. UI bude také zabudována do hotového produktu se zaměřením na uživatele. Umělá inteligence několikrát pomůže zlepšit uživatelský komfort. Bude tedy jádrem veškerého budoucího vývoje webu.

Programovací jazyky.

Tento bod se týká hlavně webových vývojářů (profesionálů i začátečníků). Jejich poptávka přímo závisí na tom, se kterými strukturami a jazyky mohou pracovat. Svou roli hraje také rychlost adaptace na nové nástroje. Proto dnes potřebují pochopit, jaké znalosti musí získat, aby udrželi krok s budoucností webového vývoje nebo ji úplně předjeli.

Javascript, Python a PHP jsou nejpopulárnější webové programovací jazyky v posledních letech[6]. Trendy se nemění, protože vývojáři jsou spokojeni s možnostmi tohoto tria. První místo nutně zaujímá Javascript, druhé je Python. A ačkoli je Python pro vývoj mobilních aplikací velmi slabý. U webových služeb založených na prohlížeči je však tento jazyk považován za jeden z nejlepších.

Současně hrají smartphony důležitou roli v našem životě. Proto je třeba věnovat stejně velkou pozornost i mobilním aplikacím. K jejich vývoji je samozřejmě vhodný Javascript. Java je však v tomto ohledu stále považována za nejefektivnější programovací jazyk.

Frameworky programovacích jazyků

Dobrá platforma je nezbytná pro dokonalou webovou službu. Součástí je vývojové prostředí. Nelze pojmenovat budoucnost o několik let dopředu, protože v každém okamžiku mohou vytvořit univerzální rámec. Ale již víme jistě, které sady nástrojů budou v příštích letech vysoce ceněny. React a Vue jsou jádrem veškerého frontend vývoje. React je dnes nejžádanější platformou ve většině IT společností. Po dlouhou dobu soutěžil s Angular. Vývojáři však tvrdí, že Angular výrazně snižuje účinnost.

Nejoblíbenější frontend frameworky

Vue se náhle stal populární v minulém roce. Převzal však asijský trh. Očekává se, že bude v souladu s React. Xiaomi je jednou ze známých značek, které zcela přešly na používání Vue.

Svelte.js je framework, který je dnes prakticky neznámý. Ale mnoho vývojářů je připraveno s ním pracovat. Je pravděpodobné, že on bude stejné populárni jako Vue. Jde o to, že Svelte vám umožňuje vytvářet aplikace, které zabírají méně paměti. Aby hotový produkt fungoval, není potřeba samotný kód frameworku. Takto je dosaženo zmenšené velikosti. Z tohoto důvodu byl Svelte pojmenován „mizí“.

Hlasový vstup

Hlasový vstup je další technologie, která zvyšuje uživatelský komfort. Proto to představuje budoucnost vývoje webu. Je jedním z příkladů implementace umělé inteligence. Dříve to nebylo tak nápadné, protože tato technologie nefungovala tak dobře. Výrobky s vysoce kvalitním hlasovým vstupem byly neúměrně drahé. Nyní je to však jeden z trendů. Hlasové vyhledávání není tak rychlé jako psaní, protože prohlížeči chvíli trvá, než jasně rozpozná řeč. Ale získává tato technologie nový vzhled a ovlivňuje vývoj většiny webových služeb. Například práce mnoha webů je založeno na hlasových dotazech. Faktem je, že v textu představujeme něco jako „zajímavá místa v Londýně“, ale řekněme „kam v Londýně?“. Navenek

docela podobné dotazy, ale jejich rozdíl výrazně ovlivňuje indexování stránek vyhledávači. Hlasové příkazy jsou také aktivně implementovány v mnoha mobilních a webových aplikacích. Uberu tedy můžete volat hlasem. Život je mnohem jednodušší, protože s aplikací nemusíte ztrácet čas. Aplikace přesně určuje, kam by měl řidič dorazit. Zbývá jen čekat na oznámení o příjezdu automobilu.

Interaktivita stránek

Nic neovlivňuje trendy vývoje webu více než uživatelská zkušenost. Koneckonců, úkolem vývojáře je vytvořit co nejpohodlnější a nejpraktičtější aplikaci. Proto budou webové služby interaktivnější. Jedním z příkladů je webová stránka autorizovaného prodejce Genesis. Místo listování stovkami stránek může uživatel jediným kliknutím prozkoumat vozidlo. Funkce navíc umožňuje „sestavit“ vaše auto, a to:

- Vybrat barvu
- zvednout kola.
- Identifikovat zařízení;
- A mnoho dalšího.

Jinými slovy, zákazník bude moci změnit vše, co nemá vliv na výkon automobilu.

Dříve nebyl takový vývoj příliš populární kvůli vysokým spotřebě zdrojů a dlouhému načítání stránek. Ale technologie je dnes rychlejší. Nyní je lepé čekat na načtení interaktivních stránek než zobrazit několik běžných.

Adaptivní design stránek

Už dávno web nenavštěvujeme pouze z počítače. Navíc stále častěji používáme smartphony a tablety, protože jejich funkčnost a výkon jsou občas mnohem vyšší. Pokud jde o používání webových aplikací, mobilní telefony jsou stejně dobré jako počítače. Jediným rozdílem je velikost obrazovky. Snadnost použití webu by neměla být ovlivněna různými monitory. V opačném případě uživatel jednoduše odmítne pracovat s webovou stránkou a přepne na lepé optimalizovanou. Každá webová aplikace se musí přizpůsobit jakémukoli zařízení, od vzhledu po funkčnost. Je důležité nevytvářet dvě (nebo více) verze webu, protože v takovém případě budou nepřesnosti patrné. Je třeba vyvinout jednu verzi, která se přizpůsobí pracovnímu prostoru uživatele.

Jednostránkové weby

Stále více vidíme na internetu jednostránkových stránek. Jakmile se takové stránky objevily, uživatelé uvedli, že to bylo kvůli lhostejnosti vývojářů k jejich projektům. Postupem času však byly výhody těchto stránek uznány. Nejprve vám umožní uložit vývojové prostředky. Za druhé, takové stránky se načítají mnohem rychleji. Moderní uživatel si opravdu cení svého času, takže nemůže dlouho bloudit po webu a hledat informace. Ještě důležitější je, že správné umístění dat na jednostránkové webové stránce vám umožní upoutat pozornost potenciálního zákazníka během několika sekund. Je mnohem pravděpodobnější, že si za 5 sekund koupí produkt, který vás zaujme, než ten, jehož studium trvalo několik hodin. Jednostránkové weby nejsou novým trendem. Ale získávají si stále větší oblibu. Implementací dalších trendů, jako je interaktivita a hlasové vyhledávání, je možně dosáhnout obrovské transformace.

Chatovací roboty a online chaty

Pokud nevíte, jak přivést ještě více zákazníků a usnadnit jim práci s aplikaci, přidejte chatbota. Jedná se o další vysoce efektivní implementaci umělé inteligence. Stejně jako jednostránkové weby, existují i roboti už po několika lety. Ale fungují jako neuronová síť. Poskytuje se jim obrovské množství dat, na jejichž základě se učí pracovat s uživatelem. Ukázalo se, že jednoduchá báze pro chatboty nestačí, protože byla vytvořena na základě FAQ. Praxe však ukázala, že FAQ uživateli velmi zřídka pomáhá, protože obsahuje zjevné odpovědi. Zákazníci častěji žádají o podporu mimořádnými otázkami.

Chatbotům chvíli trvalo, než se naučili všechny druhy chování zákazníků a zjistili, jak v dané situaci reagovat. Dnes tento proces téměř dosáhl finále. Mnoho webů má opravdu chytré chatboty. Internetové obchody tedy dnes existují bez skutečné podpory, protože je úplně nahradili roboti. Jsou připraveni skoro na jakoukoli otázku. Kromě toho si na žádost klienta může vybrat požadovaný produkt a provést objednávku. Chatboti jsou jednou z nejlepších optimalizací webových služeb. To není budoucnost, ale přítomnost. V příštích několika letech však bez nich nebude žádný projekt dokončen.

Závěr

Po dlouhou dobu bylo vytváření krásného obsahu důležitým úkolem vývoje webu. Role tohoto prvku se nezmenšila, ale vývoj technologie nám umožňuje dosáhnout vnější krásy mnohem rychleji. Celý dnešní webový vývoj musí být zaměřen na použitelnost každého prvku. Pokud je jakékoli tlačítko pro uživatele zbytečné, nemělo by se přidávat. Tím se ne jenom zvýší doba načítání webu, ale výkon se zároveň sníží.

2.2 Webové kalendáře

Jak již bylo zmíněno výše, příchod internetu do našich životů vedl k tomu, že dnes se na internetu řeší stále více úkolů. Hlavní výhodou internetu je zvýšení objemu interakcí mezi uživateli. Internet zrychlil a zjednodušil především komunikaci mezi lidmi, pomoci řady aplikací, které se objevily, což pozitivně ovlivnilo rozvoj podnikání. Jedním z příkladů takových aplikací jsou webové kalendáře.

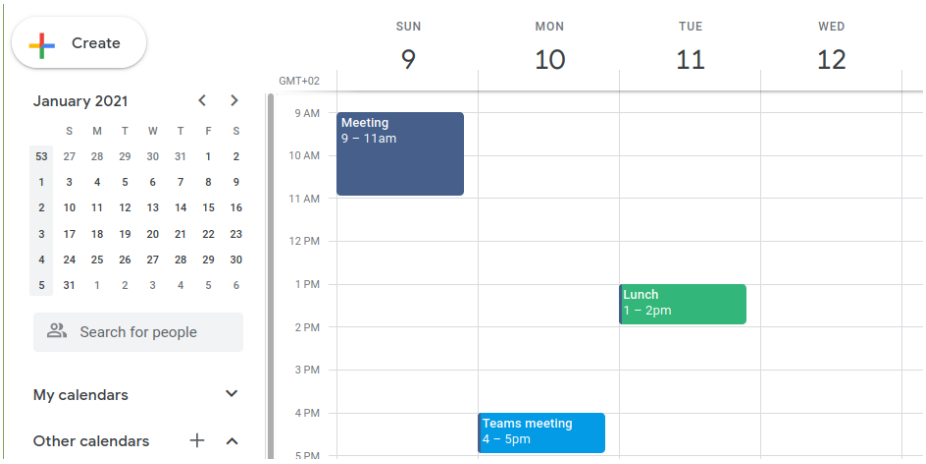
2.3 Porovnání existujících řešení webových kalendářů

V dnešní době existuje mnoho řešení webových kalendářů. Bud' jednoduché - jako jedna z komponent jiné rozsáhlejší webové aplikace nebo i všeobecně známé webové kalendáře jako samostatné aplikace. Mezi nejpopulárnější webové kalendáře jsem zvolila pro porovnání Google Calendar, Doodle a Calendly. Samozřejmě existuje i mnoho dalších realizací ale tyto tři představují nejvíc používané druhy kalendářů. Každý z nich má vlastní unikátní funkce a účely.

Google Calendar

Toto je dnes asi nejpoužívanější webový kalendář, on je poskytovaný společností Google. Tyto kalendáře jsou oblíbené u uživatelů zároveň pro pracovní i osobní použití. Každý z nás využívá pravděpodobně alespoň jeden takový kalendář. Hlavní výhodou Google kalendáře je, že poskytuje dobrou integraci mezi svými produkty. Například události v kalendáři lze

snadno přidávat, upravovat a posílat e-mailem ostatním uživatelům. Uživatelé Google také mohou v rámci jednoho účtu vytvořit mnoho kalendářů pro různé účely. Tyto kalendáře můžou také snadno sdílet a upravovat i více uživatelů spolu ve stejnou dobu. Google poskytuje nejširší sadu operací s kalendáři, které budou užitečné i v práci. Kromě toho dnes již mnoho aplikací mají integraci s Google což dělá tento kalendář univerzálním. Mezi hlavní nevýhody z hlediska jednoduchosti práce patří složitost systému jako celku. Kromě toho nemusí být vždy vhodné mít přístup ke všem svým kalendářům, protože takových kalendářů může být i příliš hodně. Z toho důvodu že téměř každý uživatel internetu má alespoň jeden účet Google, zde nebyla zohledněna potřeba registrace.



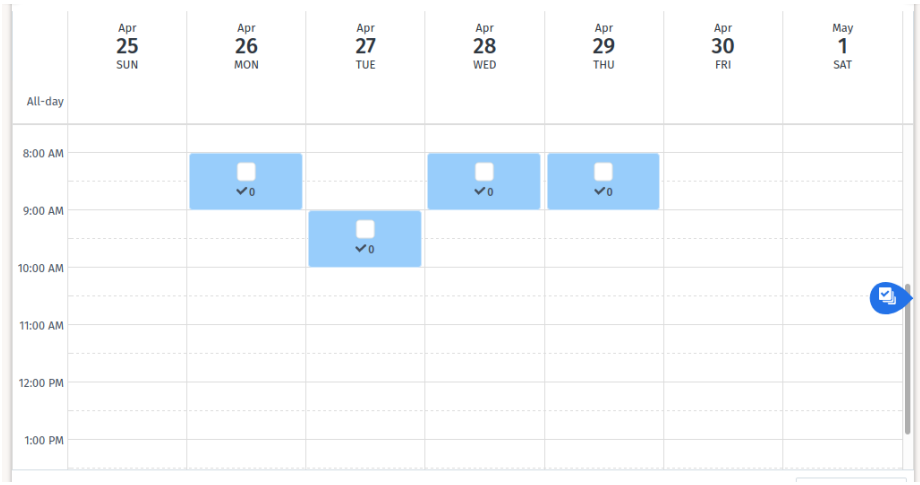
Obrázek 2.1: Uživatelské rozhraní kalendáře Google.

Doodle

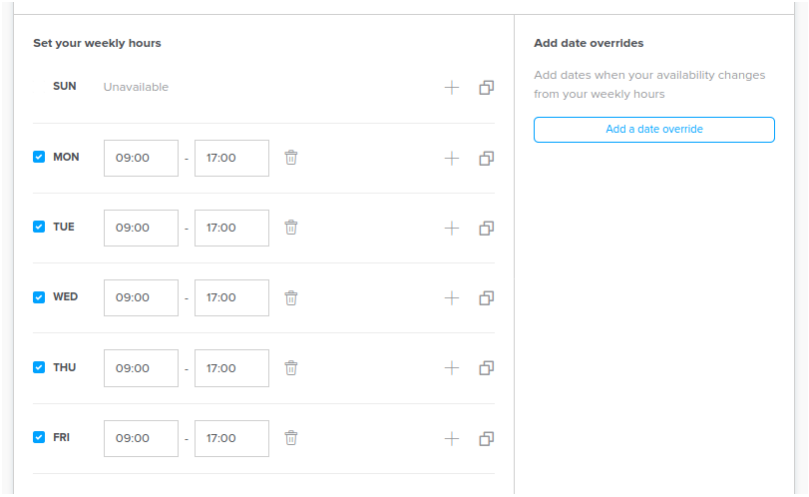
Jedná se o mírně odlišný typ kalendáře z hlediska jeho účelu a objemu funkci. Tato jednoduchá a intuitivní aplikace byla původně vytvořena pro plánování schůzek a zamezení zbytečné korespondenci. Jeden uživatel tady může vytvořit více časových úseků pro naplánovanou událost a spouštět hlasování. S pomocí tohoto hlasování si bude moci každý zvolit termín, který mu vyhovuje. Tato služba nevyžaduje, abyste na začátku prošli registrací, a umožňuje vám začít pracovat s kalendářem několika kliknutími. Pro identifikaci v kalendáři budete muset uvést pouze své jméno a e-mail. V této aplikaci však může jednotlivé termíny událostí vytvořit pouze jeden uživatel a ostatní si mohou vybrat termín pouze hlasováním, ale nemohou navrhnout své vlastní termíny.

Calendly

Třetí a poslední zde představený kalendář je Calendly. Tento kalendář je podobný kalendáři Doodle, byl také vytvořen pro plánování schůzek. Na hlavní stránce aplikace je uživateli nabídnuto projít krátkou registrací s uvedením jeho osobních údajů a e-mailové adresy účtu Google. V aplikaci Calendly může uživatel přednastavit čas, kdy je k dispozici. Ostatní uživatelé, nebo on sám, budou moci naplánovat události v jednom z těchto časových intervalů. Služba je přizpůsobena pro integraci s kalendářem Google, takže všechny události se okamžitě zobrazují v uživatelských účtech Google.



Obrázek 2.2: Uživatelské rozhraní aplikace Doodle.



Obrázek 2.3: Uživatelské rozhraní aplikace Calendly.

Ačkoli každé z představených řešení má své vlastní charakteristické funkce, nakonec jsem dospěla k závěru, že žádné z nich neodpovídá myšlence, kterou jsem vložila do svého vlastního projektu.

Kapitola 3

Návrh vlastního řešení

3.1 Analýza požadavků

Během prvních konzultaci, studia aktuálně existujících řešení a přemýšlení o předběžném návrhu byly formulovány následující funkční a nefunkční požadavky:

Funkční požadavky

1. Kalendář musí být přístupný každému uživateli přes odkaz.
2. Mělo by být možné vytvořit kalendář okamžitě, několika kliknutími bez registrace.
3. Uživatel bude moci přidávat události do kalendáře do měsíce a na týdne.
4. Události musí být možně mazat a upravovat.

Nefunkční požadavky

1. Aplikace musí běžet na technologii WebSocket.
2. Kalendář musí být přístupný prostřednictvím odkazu pro každého uživatele, takže odkaz musí být zcela unikátní.

3.2 Volba technologií

V této sekci nasleduje popis použitých technologií a kratký přehled důvodů ze kterých byli zvolené.

Backend

Jako programovací jazyk pro serverovou část aplikace jsem zvolila jazyk Python a jeho mikroframework Flask. Flask je dnes populární framework jazyka Python pro tvorbu webových aplikací. Framework je software, který zjednodušuje proces vývoje a připojení různých modulů jednoho produktu. Mikro v názvu označuje že jádro frameworku se snaží zůstávat co nejvíc lehčí, ale při tom se zachovává možnost jeho rozšíření. To znamená že sám o sobě framework „out of the box“ neobsahuje dokonce i databázovou abstraktní vrstvu, ale ale všechny potřebné komponenty můžou být přidány později ve formě knihoven nebo balíčků[10].

Flask je založen na Werkzeug knihovně pro WSGI rozhraní.

WSGI znamená *Web Server Gateway Interface*, jedná se o specifikaci, která popisuje, jak webový server komunikuje s webovými aplikacemi a jak lze tyto aplikace spojit dohromady, aby zpracovaly jeden požadavek. WSGI byla vynalezena pro standardizaci chování webových frameworků. Hlavní myšlenka teto knihovny leží v tom že cesty URL se mapují na nějakou logiku na backendu.

Místo klasického řešení využívajícího protokol HTTP byla v aplikaci použita ke komunikaci mezi klientem a serverem technologie WebSocket. Flask websocket framework poskytuje pro práce s WebSocket knihovnu Flask-SocketIO. Výhodou této knihovny je, že s jejím použitím můžeme zvolit kteroukoli z oficiálních **socket.io** klientských knihoven nebo libovolného jiného kompatibilního klienta k navázání spojení[8].

Python

Programovací jazyk Python je známý svou jednoduchostí a stručností. Stručná a jasná syntaxe podobná pseudokódu a silné dynamické psaní přispívají k rychlému a bezbolestnému učení nováčků.

Interpret jazyka se postará o veškerou práci na nízké úrovni a osvobodí programátora od nutnosti manuální správy paměti. Praktická nemožnost získání poruchy segmentace, stejně jako pohodlný systém výjimek dodávaný s jasnými zprávami, vám umožní rychle ladit programy. Situace, kdy jejich selhání kvůli chybě, ke které došlo, vyžadují hluboké ladění, jsou poměrně vzácné. Nepřeplněná celá čísla a bezpečnost používání standardních knihovních kontejnerů dělají z Pythonu dobrý pre-prototypovací nástroj pro nápady a díky velkému počtu vysoce kvalitních matematických knihoven je lídrem v oblasti i strojového učení, analýzy dat a vědeckých výpočtů.

Sofistikovanější programátoři oceňují tento jazyk pro jeho pohodlné nástroje pro vytváření *pipelinů* odložených nebo, jak se říká, líných výpočtů. V Pythonu je tato funkce implementována iterátory a tzv. generátory. Knihovna asynchronního programování je také docela dobrá.

Ale Python má také své nevýhody. Za hlavní z nich je považována jeho pomalost, i když tomu čelí určitá míra spravedlnosti skriptovací jazyk ve skutečnosti nepotřebuje rychlost. V úlohách vyžadujících vysoký výkon funguje pouze jako obal pro manipulaci s API nízkoúrovňových knihoven napsaných v jazycích s podporou AOT kompilace (Ahead-of-Time). Nejpopulárnější z těchto jazyků jsou samozřejmě C a C ++. První například implementuje široce používanou knihovnu `NumPy` vytvořenou pro matematické operace s poli libovolných rozměrů. Na druhé - stále populárnější rámec pro výcvik neuronových sítí `PyTorch`.

Ať je to jakkoli, v čistém Pythonu nebudete moci napsat něco vysoce výkonného. Proto se vyžaduje uchýlení k pomoci jiných jazyků nebo použití staticky zadaných přípon, jako je například `Cython`, do kterého je, mírně řečeno, nepříjemné psát. Nedostatek rychlosti obecně omezuje rozsah tohoto jazyka na úkoly, ve kterých je čekací doba na odpověď na požadavek mnohonásobně delší než doba běhu těla skriptu[1].

Porovnaní protokolů WebSocket a HTTP

HTTP *(Hyper Text Transfer protocol)* protokol je založen na principu dotaz - odpověď. Klient zasílá požadavky a server na každý požadavek zasílá odpověď která se v podstatě skládá ze dvou části - příslušné hlavičky a dat - těla odpovědi. Spojení se tak navazuje pro každý požadavek zvlášť a klient zasila je bez ohledu na to jeli dostupna odpověď' od serveru. Protokol WebSocket na rozdíl od HTTP protokolu poskytuje plně duplexní, obousměrný

komunikační kanál který využívá jediné TCP spojení. Navázání spojení u WebSocketů probíhá jenom jednou na začátku komunikace. Toto spojení se udržuje po celou dobu dokud nebude přerušeno nebo uzavřené. Klient a server tak můžou volně zasílat zprávy mezi sebou v libovolný okamžik, server nemusí čekat na požadavek od klienta. Takový způsob zrychluje komunikace a děla je efektivnější[11].

Formát zpráv

Jako formát zprávy pro komunikaci mezi klientem a serverem byl vybrán JSON (JavaScript Object Notation). Jedná se o lehký a snadno srozumitelný formát pro výměnu dat. JSON představuje data v textové podobě jako páry klíč – hodnota a podporuje libovolný počet úrovní vnoření.

Díky své jednoduchosti se JSON stal populárním formátem a nyní ho většina programovacích jazyků podporuje přímo nebo prostřednictvím knihoven. Některé relační databáze, jako jsou PostgreSQL a MySQL, nyní navíc podporují třídění a dotazování dat přímo ve formátu JSON[3].

Frontend

Klientská část aplikace byla plně implementována v jazyce JavaScript. JavaScript je moderní, dynamicky se rozvíjející skriptovací programovací jazyk, který se dnes používá pro tvorbu webových aplikaci, je objektově orientovaný a multiplatformní. Kromě jednoduché syntaxe a jednoduchého jazykového modelu je hlavní výhodou JavaScriptu obrovské množství knihoven, které lze použít k implementaci jakéhokoli řešení. JavaScript se spouští a běží úplně v prohlížeči, což zaručuje multiplatformnost.

Aplikace však není napsána v čistém JavaScriptu, ale pomocí jeho knihovny React a jedná se o tzv. SPA (Single Page Application).

Výhody a nevýhody JavaScript

Stejně jako všechny počítačové jazyky má JavaScript určité výhody a nevýhody[9]. Výhody JavaScript

- Rychlý pro koncového uživatele: kód jazyka JavaScript je napsán na straně klienta, není nutná žádná podpora webového serveru. Také to nemusí být kompilováno na straně klienta, což mu dává určité rychlostní výhody. Protože se skript provádí v počítači uživatele, v závislosti na úkolu jsou výsledky téměř okamžité. Před odesláním požadavku na server můžete například ověřit jakýkoli vstup uživatele. To snižuje zátěž serveru.

- Jednoduchost: JavaScript je relativně snadné se naučit a implementovat. Využívá DOM, který poskytuje mnoho předdefinovaných funkcí pro různé objekty na stránkách, což usnadňuje vývoj skriptu, který by vyhovoval vlastnímu účelu.

- Všestrannost: JavaScript funguje skvěle s jinými jazyky a lze jej použít v široké škále aplikací. V současné době existuje mnoho způsobů, jak používat JavaScript prostřednictvím serverů `Node.js`.

Nevýhody JavaScript:

- Zabezpečení: JavaScript je přímo přidán k webovým stránkám a klientským prohlížečům, může zneužít systém uživatele, takže na klientském počítači lze spustit škodlivý kód.
- Podpora prohlížeče: JavaScript je někdy interpretován odlišně různými prohlížeči. Různé mechanizmy rozložení mohou vykreslovat JavaScript různými způsoby, což vede k nekonzistenci, pokud jde o funkčnost a rozhraní. Velká část JavaScriptu závisí na manipulaci s prvky DOM pomocí prohlížečů. A různé prohlížeče poskytují různé typy přístupu k objektům.
- Stále více konkurentů: JavaScript je velmi starý skriptovací jazyk, který běží na strojích a existují již i jiné technologie, které dělají totéž (jako JQuery) lepším a snadnějším způsobem.
- Možnost zakázat JavaScript: Pokud v svém prohlížeči zakážete JavaScript, žádný kód JavaScript se nespustí.
- Nahrání souboru: Soubor JavaScriptu je načten na klientském počítači, aby si každý mohl přečíst kód a znovu jej použít.

DOM

DOM (Document Object Model) je to způsob, reprezentace obsahu dokumentu HTML jako objekty. Navíc existuje rozhraní pro správu zadaných objektů. DOM je rozdělen na běžný DOM (nazývaný také skutečný) a virtuální.

Skutečný model DOM bude aktualizovat všechny tagy, dokud nenajde požadovaný fragment. A to v některých případech negativně ovlivňuje výkon a další parametry. Virtuální DOM aktualizuje pouze požadovaný blok HTML.

React

React[4] je knihovna jazyka JavaScript pro vytváření webových rozhraní, navržená společností Facebook v roce 2011. React umožňuje vývojářům neinteragovat přímo s DOM (Document Object Model), ale místo toho vytvoří další abstraktní vrstvu React DOM.

React navíc také zjednodušuje samotný vývoj. Jednotlivé části aplikace lze vytvořit jako komponenty, které lze později snadně kombinovat do jednoho celku. Pracuje s webovými a nativními aplikacemi, pro jejichž vývoj využívá React Native a pro vývoj multiplatformních aplikací React Native Renderer.

Na závěr můžeme říct že React má následující výhody:

- Používá DOM.
- Aplikaci lze snadno rozšířit
- Dokáže odolat těžkým nákladům
- Je založen na jednoduchých programovacích jazycích.
- Má malou váhu dat
- Má otevřenou knihovnu

- Podporuje snadnou migraci mezi verzemi
- Kód uživatelského rozhraní je čitelný a snadno se udržuje

Kromě všech výhod má React také některé známé nevýhody.

- React místo HTML používá JSX.
- Dokumentace není moc dobrá. To může na začátku studie Reactu pro některé programátory představovat problém.
- React je obvykle potřeba k vytvoření uživatelského rozhraní. K tomu jsou potřebné knihoven třetích stran.

Základní stavební kameny aplikace v Reactu se nazývají „komponenty“. Dříve byly komponenty objekty tříd, i když samotné třídy v jazyce JavaScript jsou pouze syntaktickým cukrem a byly do jazyka přidány již později. V moderní verzi React je komponenta funkce napsaná v kódu JavaScript, ale zároveň musí vracet nějaký kód ve formátu podobném označení html. Toto označení není čistý html, ale nějaká kombinace HTML a JavaScript.

JSX

JSX je zkratka pro *JavaScript XML*. Toto je dodatek k syntaxi JavaScriptu. Používá se k psaní kódu podobného XML pro prvky a komponenty uživatelského rozhraní [5]. Kód JSX je podobný XML v tom, že má také uzly, tyto uzly mají názvy, hodnoty a atributy. Tento kód je zcela spuštěn uvnitř souboru JavaScript a dokonce i samotná komponenta má příponu `.js`, takže můžeme snadno spustit kód JS přímo uvnitř komponenty JSX, pro tento účel musíme tento kód vložit do složených závorek.

JSX je speciální typ označení, který byl vytvořen vývojáři React. Jak je popsáno výše, React nepracuje přímo s DOM, ale vytváří další abstraktní vrstvu. Proto i přišli tvůrci Reactu s JSX. Toto označení je podobné HTML, ale je určeno pro práci s React DOM. Syntaxe tohoto označení se však od html liší, protože se stále jedná o kód JavaScript a, například zde bylo nahrazeno klíčové slovo `class` klíčovým slovem `className`, aby nedocházelo ke konfliktům. Navíc samotný React DOM používá pro pojmenovaní vlastnosti objektů místo obvyklých názvů atributů HTML typ camelCase. I když je možné psát aplikaci vůbec bez JSX, jeho použití přesto děla kód jednodušším, čitelnějším a elegantnějším.

SPA

Dříve se komunikace mezi klientem a serverem skládala z následujících fází:

1. Inicializační požadavek od klienta na server
2. V reakci na tento požadavek server generoval a odesílal často velký HTML soubor, který obsahoval všechny informace, které bude v budoucnu klient zobrazovat.
3. Poté pro každém POST požadavku od klienta server znovu posílal celý HTML soubor, který nyní obsahoval navíc i požadované informace, ale tyto informace mohly stanovit pouze malou část původního HTML souboru.

Zatímco aplikace SPA načtou HTML stránku pouze jednou. Malý HTML soubor přichází jako odpověď ze serveru pouze na inicializační požadavek a poté se stránka nikdy úplně znovu nenačte. Klient odesílá požadavky a server vrací pouze soubor, obvykle ve formátu JSON, který obsahuje jenom informace, které klientovi chybí. A ačkoli tento soubor může kvůli svému formátu obsahovat velké množství informací, zůstává relativně malý. Možnost nestahovat neustálé HTML souboru nám umožňuje ušetřit množství dat a čas potřebný k jejich přenosu. Tento přístup výrazně snižuje zátěž serveru.

Toto je princip Single Page Application. Máme pouze jednu HTML stránku, ale JavaScript kód na této stránce se postará o odesílání požadavků, zpracování a vykreslování dat přijatých ze serveru. Pokud byl dříve pro každou záložku webu potřeba samostatný HTML soubor, pak díky knihovně React máme vždy jen jednu relativně malou HTML stránku, která přijímá spoustu dat v JavaScriptu. Pokud chceme v aplikaci SPA přejít na novou kartu, nový HTML se nenačte ale JavaScript kód jednoduše zobrazí potřebná data.

Server se tedy změní na *web api (web application programing interface)*. Protože nyní neposílá data ve formátu HTML, ale v nějakém univerzálním formátu (XML nebo JSON), klient teď nemusí být nutně webová stránka, ale v podstatě samostatná webová aplikace, která má vlastní logiku.

React State a React Hooks

Klíčovou výhodou jakékoli aplikace Reactu je použití takzvaných „háčků“(Hooks). Oni jsou a v tomto případě hlavním přínosem této knihovny pro danou aplikaci. Hooks umožnily implementovat mnoho funkcí, které by bez nich vyžadovaly poměrně velké množství kódu. To byl hlavní důvod, proč jsem se nakonec rozhodla napsat frontend aplikace v Reactu. Následuje krátký přehled této skvělé vlastnosti Reactu, protože si zaslouží pozornost.

Smyslem React Components je pracovat s jeho stavem. Každá komponenta má svůj vlastní vnitřní stav a pokud se on změní, tak React překreslí přímo oblast, kde k této změně došlo. Tento stav byl dříve obsažen ve třídě, kterou dědily všechny komponenty. Ale později byl React přenesen na funkční komponenty, protože za prvé je to jednodušší a za druhé, šetří to zdroje, protože nyní nepotřebujeme dědit z několika entit. Při přechodu na funkční komponenty bylo nutné zachovat interakci se stavem, který dříve pohodlně zajišťovali komponenty třídy.

Interakce se stavem komponenty zahrnuje především práci s interním stavem komponenty a s metodami jejího životního cyklu, jako je `ComponentDidMount`, `ComponentDidUpdate` a další. To vše fungovalo se součástmi třídy, protože na základě třídy React v paměti vytvářel objekt a poté s tímto objektem pracoval. Objekt měl také i své vlastní metody. Pravě proto byly vynalezeny Hooks - aby nahradili tyto nutné operace u funkcionálních komponent.

Hooks jsou oddělené izolované funkce, které vám umožňují implementovat všechny základní funkce Reactu pro práci s komponentou. Použitím hooků zůstane kód rychlejší a schopny k škálovaní. Měli byste však používat háčky opatrně, protože pokaždé, když zavoláte funkci obsahující hook, React si zapamatuje toto volání a stav komponenty a potom neuloží ho do objektu, jako tomu bylo dříve, ale zvlášť do jiného místa, takže „lokální“ stav nemusí být docela vhodný název. Níže je‘uveden popis některých základních hooků Reactu.

UseState

Toto je možná nejčastěji používaný hook a tato aplikace není výjimkou. UseState je, jak název napovídá, funkce odkazující na nějaký místní stav. Syntaxe pro UseState je následující:

```
const [value, setValue] = useState(initial_value);
```

Funkce UseState vrací n-tici (tuple). Tuple je pole, jehož obsah je předdefinován. První prvek pole vráceného funkcí useState je samotný stav a druhý prvek je funkce, která umožňuje tento stav změnit tak, aby React mohl vidět, že se stav opravdu změnil, a následně překreslit šablonu. Později tedy můžeme přistupovat ke stavu uloženému v hodnotové proměnné a také jej měnit pomocí funkce `setValue`. UseState je užitečný pro dynamické zobrazování dat na stránce - jakmile se stav změní, obrazovka se okamžitě překreslí.

UseEffect

Jedná se o hook, který umožňuje sledovat změny stavu. Příklad syntaxe UseEffect je následující:

```
useEffect( () =>  /*some code*/ , dependencies)
```

Uvnitř složených závorek je napsán kód, který musí být proveden, když je splněna určitá podmínka. Samotné podmínky jsou zapsány do pole závislostí (dependencies).

Vizuální reprezentace

Implementace stylů a animací komponent v projektu může být dlouhý a složitý proces. Kromě toho není vždy tak snadné udělat web opravdu atraktivním. Z tohoto důvodu byla většina vizuální prezentace implementována pomocí existujících knihoven. Tyto knihovny mají hotová docela pěkná designová řešení a otevřený zdrojový kód. Většina vizuálních prvků, stejně jako samotné označení stránky, jsou již hotové komponenty jako součást odpovídající knihovny. V tomto projektu byly použity dvě takové knihovny.

Bootstrap

Bootstrap je podle jeho webových stránek nejpopulárnější HTML, CSS a JS framework pro tvorbu responzivního webu. Bootstrap je sada nástrojů pro rychlejší a snadnější tvorbu takových stránek. Tato knihovna, která obsahuje hotové CSS a javascriptové části kódu, díky nimž můžete rychle vytvářet responzivní a mobilní webové stránky. Responzivní web je web, který automaticky upravuje své rozložení tak, aby odpovídalo obrazovce vašeho zařízení, od velkých monitorů po telefony. Přizpůsobivost Bootstrapu je dána jeho modulární sítí. Tato mřížka se přizpůsobí šířce obrazovky a obsahuje 12x12 položek. Bootstrap obsahuje mnoho komponent pro interakci s uživatelem, jako jsou tlačítka, formuláře, vyskakovací okna a další. Přidat Bootstrap na stránku můžete připojením příslušných JavaScript a CSS souborů.

Material UI

Jedná se o speciální samostatnou knihovnu, sadu komponent React, která implementuje Material design Google. Komponenta Material UI pracuje izolovaně. To znamená, že je sa-

mostatná a zavádí pouze styly, které by měla zobrazovat. Podobně jako Bootstrap obsahuje Material UI mnoho předdefinovaných komponent. Tyto komponenty jsou kombinací kódu JavaScript, JSX a CSS. Umožní vám vytvářet krásné webové stránky s elegantními styly a dokonce i animacemi. Díky uživatelskému rozhraní v Materiál UI bude váš web vypadat profesionálněji.

Databázový systém

Jako databázi pro svou aplikaci jsem si vybrala relační databázi PostgreSQL. Relační databáze je založena na relačním modelu. Data v takové databázi jsou často prezentována ve formě tabulek jako sada vztahů. Relační model je logický model, data v něm jsou prezentována jednotně a jsou spojena odkazy. A i když existuje mnoho různých SŘBD (Systémů řízení báze dat), všechny se řídí stejnými pravidly a se všemi je možné manipulovat pomocí jazyka SQL. Databáze PostgreSQL je pro danou aplikaci velmi vhodná. Zvládne zátěž, když mnoho uživatelů budou současně zapisovat a číst data z různých tabulek.

Práce s databází na serveru se neprovádí přímo pomocí dotazů, ale pomocí takzvaného ORM SQLAlchemy.

SQLAlchemy

Při vývoji aplikace nemusí vývojář vědět, která databáze bude použita, nebo může být v procesu vybrána i jiná databáze. Proto je nutné vytvořit univerzální program, který by nebyl vázán na konkrétní SŘBD. V Pythonu se k tomu často používá SQLAlchemy. SQLAlchemy je rozhraní postavené na technologii ORM (Object relational mapping) pro práci s databázemi v jazyce Python. ORM je technologie pro překlad dat mezi systémy s nekompatibilními datovými typu pomocí objektově orientovaného jazyka. SQLAlchemy vám umožňuje pracovat s databázovými tabulkami jako s objekty Pythonu. Toto rozhraní je navíc univerzální a na úrovni WSGI (Web Server Gateway Interface) aplikace není vázána na konkrétní SŘBD. To znamená, že stačí nastavit spojení s SQLAlchemy a pak můžeme pracovat s libovolnou databází beze změny kódu samotné aplikace.

Flask SQLAlchemy je postaven na SQLAlchemy. Práce s databázemi probíhá prostřednictvím speciální třídy. Jednotlivé databázové tabulky jsou zde také reprezentovány jako třídy Pythonu. Třídy mají pole to jsou sloupce tabulky. V polích lze určit vlastnosti i omezení jejich hodnot.

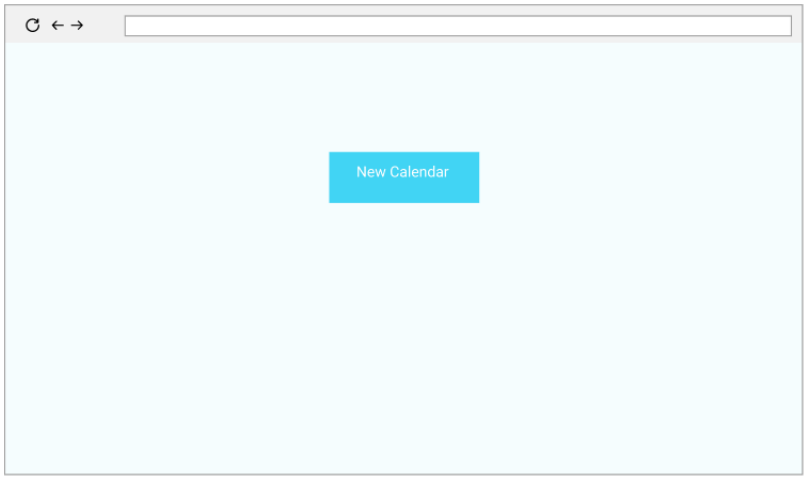
3.3 Návrh uživatelského rozhraní

Aplikace měla obsahovat pouze 3 stránky.

- Úvodní stránka s minimalistickým designem a tlačítkem vyzývajícím k vytvoření kalendáře
- Stránka s měsíčním kalendářem, na které by bylo možné vytvářet události, které nejsou vázány na konkrétní čas, ale obsahují informace pouze o datu.
- Stránka s týdenním kalendářem, který by zobrazoval termíny během jediného dne.

K návrhu stránek jsem použila aplikace moqups.coma Figma. Tyto aplikace umožňují uživateli vytvářet návrhy webových stránek a generovat pro ně CSS styly.

Úvodní stránka aplikace



Obrázek 3.1: Úvodní stránka aplikace.

Hlavní stránka neobsahuje nic zbytečného kromě jediného tlačítka, aby si uživatel mohl rychle vytvořit nový kalendář. Po kliknutí na toto tlačítko bude vygenerován nový kalendář s unikátním ID a přesměruje uživatele na stránku na nové kartě obsahující měsíční kalendář.

Měsíční kalendář

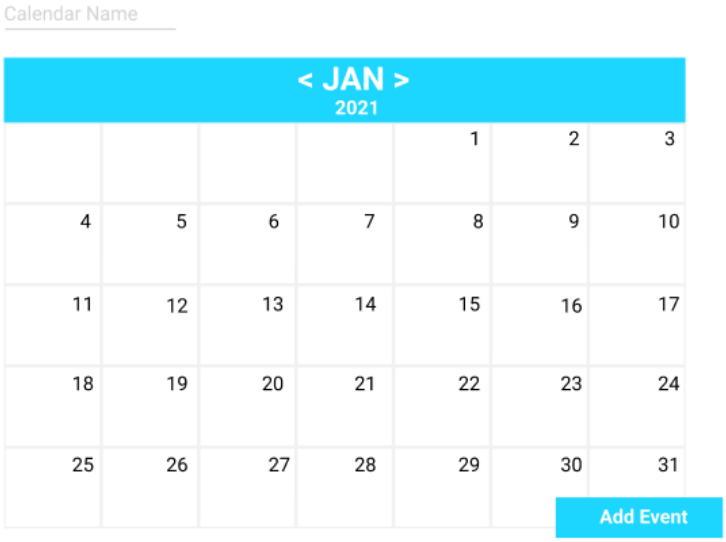
Měsíční kalendář obsahuje klasický vzhled na 1 měsíc. Stránka v zásadě se skládá ze dvou částí:

1. Název kalendáře - pole pro název kalendáře.
2. Samotný kalendář.

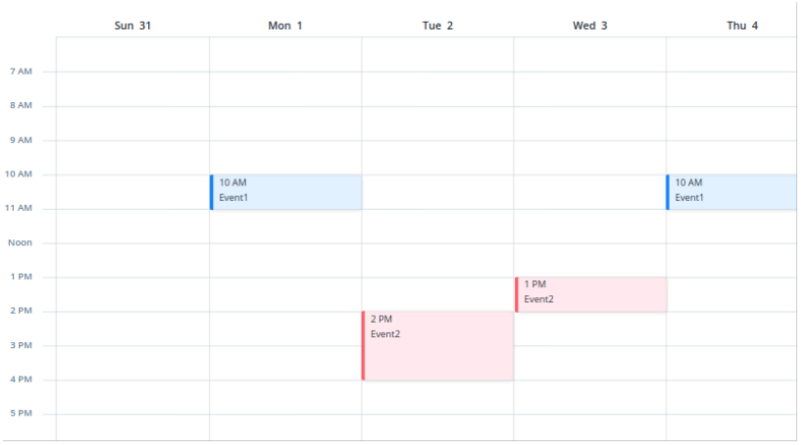
Na stránce je také tlačítko pro přidání události kliknutím na které se otevře vyskakovací okno s formulářem.

Týdenní kalendář

Týdenní kalendář je podobný jiným existujícím řešením a obsahuje sedm dní v týdnu s buňkami pro hodiny. Události z kalendáře se zobrazují podle dne v týdnu jako obdélníky. Každý takový obdélník obsahuje čas zahájení události a název. Na stránce je také k dispozici kontextové menu. Kliknutím pravým tlačítkem myši na událost ji můžete upravit nebo smazat.



Obrázek 3.2: Stránka měsíčního kalendáře.



Obrázek 3.3: Stránka Týdenního kalendáře.

Kapitola 4

Implementace

Aplikace se skládá ze dvou částí - klienta a serveru. Tyto programy se spouštějí a běží současně na dvou různých portech.

4.1 Backend

Flask, stejně jako mnoho jiných webových frameworků, vyžaduje organizaci aplikačních modulů podle nějakého vzoru. Tato šablona umožňuje organizovat a strukturovat soubory programu.

Struktura aplikace Flask

Ve Flasku jsou všechny moduly rozděleny do balíčků. Aby bylo možné aplikaci spustit, je třeba vytvořit jeden hlavní balíček. Balík v pythonu je v podstatě adresář, který obsahuje soubor `__init__.py`. To znamená, že každý adresář, který obsahuje soubor `__init__.py`, bude považován pythonem za balíček. Použití balíčků je způsob, jak strukturovat jmenný prostor modulu. Dva balíky tak mohou obsahovat moduly se stejným názvem[7]. Obecně je Flask velmi lehký a můžete spustit samostatnou nezávislou aplikaci napsáním několika řádků kódu. Ve vývoji softwaru je ale dobré rozdělit program do samostatných modulů podle funkcí, které oni vykonávají.

Z toho důvodu že pracuji s kalendářem, funkce pro zobrazení týdne se příliš neliší od funkcí pro zobrazovaní měsíce, snažila jsem se je přivést do nejuniverzálnější podoby a obecně jsem se hodně soustředila n refaktoringu kódu, aby byl co nejuniverzálnější. To nejen zlepšuje čitelnost kódu, ale opravdu usnadňuje vývoj. Nakonec tak vývojář vždy ví, jakou konkrétní funkci v tuto chvíli potřebuje. Vzhledem k tomu, že jsou funkce univerzální, lze je také snadno kombinovat a výstupy z jedné funkce lze jednoduše přijímat jako vstup jinou funkcí.

O zpracovaní kalendáře stará balík `calendars`. Samotné funkce se nacházejí v souboru `controllers.py`.

Websockety

Vzhledem k tomu, že websockety vždy odesílají pouze textové informace ve zprávách a pouze takové množství dat, které je skutečně potřeba, může být rychlost přenosu zpráv udržována velmi vysoká i při velkém objemu těchto zpráv. Hned na začátku vytváření mé aplikace jsem však narazila na problém broadcastu. Broadcast u websocketů znamená že zprava

od serveru bude poslaná najednou všem klientům, což nebylo by vhodné pro soukromé kalendáře. V takovém případě má Flask WebSockets řešení - *roomky (rooms)*. Roomky vám umožňují zúžit kanál posílaní zpráv. Server se pomocí funkce join připojí k roomce, ke které se mohou klienti také připojit, a poté se přenos zprávy uskutečni pouze v této roomce a nebude odeslán klientům, pro které není určen. Aby byl identifikátor místnosti jedinečný, ale zároveň nebylo nutné generovat spoustu nepotřebných dat, bylo jako takový identifikátor zvoleno ID kalendáře. Velkou jedinečnou posloupnost čísel lze tedy vygenerovat pouze jednou - při vytváření kalendáře.

Vytváření kalendáře

Jak to bylo napsáno v návrhu řešení, hlavní stránka obsahuje pouze jedno tlačítko, po kliknutí na něj se vygeneruje nový kalendář. Samotné vytváření probíhá určité na serveru. Po kliknutí na tlačítko klient odešle zprávu do websocketu obsahující požadavek na vytvoření kalendáře. Po obdržení této zprávy server generuje pomocí funkce `uuid.uuid4()` jedinečné šestnáctkové číslo o délce 32 znaků. Tato funkce byla vybrána, protože generuje sekvence dostatečně vysoké úrovně složitosti, což znamená, že kalendář bude chráněn docela dobře.

Vygenerovaná sekvence se zapíše do databáze jako pole obsahující primární klíč nově vytvořený položkami tabulky `Calendars`. K přidání nového kalendáře do databáze však dojde později. To bylo provedeno za účelem zjednodušení kódu, aby byl univerzálnější. Po vytvoření tohoto jedinečného identifikátoru bude funkce vygeneruje odpověď která teď bude obsahovat ID nového kalendáře, který bude klientem použit k vykreslovaní stránky měsíčního kalendáře. Jak bylo napsáno v předchozí kapitole po vygenerovaní id, nový kalendář se otevře na nové kartě, kde se načte jeho stránka. Zde klient znovu odešle ID na server, aby získal data z existujícího kalendáře. A teprve potom, pokud kalendář s takovým identifikátorem v databázi neexistuje, bude vytvořen. V odpovědi na tuto zprávu místo událostí z kalendáře, které ještě nejsou k dispozici, jednoduše přijde prázdné pole.

Měsíční kalendář

Měsíční kalendář sestává z několika bloků, za kterými vždy je funkce, která zpracovává požadavky související s tímto blokem.

Prvním blokem je pole názvu kalendáře, které původně obsahuje zástupný symbol „Nový kalendář“ jako implicitní hodnotu. Toto pole v podstatě je forma, o zpracování které také se stará server. Toto pole v tabulce již obsahuje hodnotu, kterou server obdrží z databáze a odešle klientovi uvnitř funkce `GetCalendarName`. Přidání názvu kalendáře v podstatě je aktualizace její hodnoty, která bude probíhat stejným způsobem, jako se aktualizují ostatní hodnoty v celé aplikaci. Samotná změna probíhá ve funkci ChangeCalendarName. Server obdrží jako argument id kalendáře a jednoduše nahradí staré jméno novým.

Druhou velkou položkou na stránce je samotný kalendář. Nakreslí a postará se o většinu souvisejících operací strana klienta, a proto bude tento blok popsán později.

Třetí a poslední položka na stránce je seznam událostí. Zde se zobrazují pouze události, které nejsou časově omezené, což znamená, že trvají několik dní. Klient je také zobrazuje, ale zde by bylo vhodně popsat, jak se tyto data dostanou ke klientovi.

Na straně serveru je funkce, která je volána v reakci na zprávu od klienta. Tato zpráva přichází vždy při načtení stránky a poté pokaždé, když klient potřebuje data z databáze. Funkce, která zpracovává, tuto zprávu dostane na vstup ID kalendáře. Dále pomoci tohoto ID vybere všechny události, které patří do daného kalendáře, a sebere z nich JSON zprávu

pro klienta. Tato zpráva má obecný formát a klíče z párů klíč – hodnota jsou univerzální, takže ji mohou přijímat i další funkce.

Přidávání událostí

Jednou z hlavních funkcí kalendáře je schopnost přidávat události. Tato funkce se volá podobným způsobem odkudkoli v mé aplikaci. Navzdory tomu, že klient obsahuje dva formuláře pro přidání události, které budou popsány níže, přicházejí tyto požadavky na server stejně a server také odesílá jednotná data jako odpověď.

Funkce `addEventForm` se stará o přidávání událostí. Přijímá data z formuláře události přidání na klientovi. Jako vstupní parametry přijímá tato funkce zprávu JSON, která obsahuje data zadaná uživatelem a serializované. Protože tentokrát již funkce id nevyžaduje, může jednoduše vytvořit novou položku Tabulky událostí. Do tabulky budou tedy uloženy všechny poslané údaje. Jako odpověď funkce znovu provede dotaz v tabulce s událostmi a podle čísla kalendáře vrátí všechny záznamy patřící k tomuto kalendáři, včetně nově přidané události.

Rušeni události

Proces mazání události probíhá ve funkci mazání. Tato funkce bere jako vstupní parametr ID události, která má být odstraněna. Vzhledem k tomu, že nyní obsahuje jenom potřebný id, můžete snadno a rychle odstranit požadovanou položku z tabulky Calendars, protože tabulka `Events` obsahuje všechny události dohromady. Žádné pomocné tabulky nejsou potřeba, protože neexistuje žádný vztah n-ku-n. Jako odpověď se znovu vrátí kompletní seznam událostí souvisejících s tímto kalendářem, tento seznam bude později použit klientem.

Aktualizace události

Aktualizaci události, tj. změnu již přidané do databáze hodnoty, provádí server ve funkci `updateEvent` Tato funkce také přijme zprávu od klienta ve stejném formátu jako i funkce pro přidání nové události. Tato zpráva obsahuje všechna data související s touto událostí ze dvou důvodů. Prvním důvodem je, že o události není mnoho údajů a nemá smysl jich omezovat. Druhým důvodem, proč se přenášejí všechna data, je to, že tento přistup umožňuje funkci být univerzální a nezáviset na tom, která pole chce uživatel aktualizovat.

Jinak tato funkce funguje docela intuitivně - podle id výběre potřebný záznamu z databáze a přepíše aktualizované hodnoty. Jako odpověď, jako vždy, přijde celý seznam událostí.

Tím je ukončen popis klíčových funkcí implementovaných na straně serveru aplikace. Jak vidíte, Flask nám umožňuje vytvářet malé, jednoduché a univerzální funkce. Díky tomu si i často rozsáhlá aplikace muže udržovat jednoduchou a snadno srozumitelnou strukturu.

Websockety na klientu a zasílání zpráv Samotna websocket vrstva se také skládá ze dvou částí. Stejně jako aplikace websocket má spojené také dvě části - server a klient. A pokud na serveru běží Flask SocketIO server, pak na klientu **socket.io**.

SocketIO je knihovna, která poskytuje přístup ke komunikaci na vrstvě websocketů a lze ji použít na serveru i na klientovi. Protože komunikace SocketIO je tzv. „založená na událostech“, zprávy mezi klientem a serverem se také přenášejí jako události.

Protože je na backendu pro každou z operací implementována pouze jedna funkce, klient také vysílá pouze několik typů zpráv v souladu s funkcemi na straně serveru, které tyto zprávy přijímají a obsluhují.

Proto, jak již bylo popsáno dříve, se veškerá práce klienta scvrkává na zpracování dat a jejich redukci do formy, kterou lze přenášet v jedné z websocket událostí pomocí funkce `emit`.

SocketIO je knihovna, která poskytuje přístup ke komunikaci websocket a lze ji použít na serveru i na klientovi. Protože komunikace SocketIO je tzv. Založená na událostech, zprávy mezi klientem a serverem se také přenášejí jako události.

4.2 Frontend

Jak je popsáno v předchozí sekci, klientská strana aplikace byla napsána pomocí knihovny React jazyka JavaScript a byla implementována jako jednostránková aplikace. Jak naznačuje slovo Aplikace, nemluvíme o klientovi, který neprezentuje jednoduše data, která přijímá. Tady mluvíme o plnohodnotné aplikaci, která má svou vlastní logiku a může samostatně provádět velké množství operací.

Aplikaci React také tvoří moduly - takzvané komponenty. Tyto komponenty jsou logicky rozděleny, tentokrát do samostatných adresářů.

Klient je rozdělen na jednotlivé součásti, pro každou z nich je přidělen vlastní adresář. Aplikace obsahuje následující adresáře:

- Kalendář
- Události
- Týden

Adresář Calendar obsahuje všechny komponenty související s měsíčním kalendářem. Tyto komponenty odpovídají hlavním vizuálním prvkům na stránce.

Vykreslovaní kalendáře

Všechny vizuální prvky související s měsíčním zobrazením kalendáře jsou umístěny v kořenové složce `Month.js`. Zbytek komponent jsou její potomci. Hlavní část procesu vykreslování kalendáře se nachází uvnitř komponenty `Calendar.js`. Obsahuje pohledy na samotný kalendář a formulář pro přidání nebo aktualizaci události, stejně jako i formulář s názvem kalendáře.

Data pro samotný kalendář se započítávají a poskytují pomocí funkce `render`. Aktuální datum bere jako vstupní parametr. Po načtení stránky se kalendář vždy otevře v aktuálním datu. Funkce vrací pole, které bude předáno zpět komponentě. Tato hodnota bude zapsána do stavu komponenty a vrácena návratovým blokem jako kód JSX. Dostaneme tedy na stránku v zásadě tabulku s daty požadovaného měsíce.

Zobrazení seznamu událostí na stránce měsíce

Seznam událostí je jednotný pro celý kalendář, aby událost bylo možné snadno najít a pro potřebnou nám mohli prohlížet detaily.

Tento seznam se vykresluje pomocí funkce `renderEventsList`. Tato funkce přijímá jako vstup pole, které klient obdržel ze serveru. Způsob, jakým klient přijímá data, si zaslouží zvláštní pozornost a bude popsán níže v příslušné části. Ale pro tuto chvíli funkce `renderEventsList` transformuje přijatá data do požadovaného formátu. Je třeba říci, že na rozdíl od serveru se formáty výsledných polí na klientovi již navzájem výrazně liší,

proto ve skutečnosti musíme pro každou komponentu vygenerovat vlastní samostatné pole. Toto pole bude vráceno komponentě Calendar a umístěno na stránku pomocí funkcí `map` jazyka JavaScript jako samostatné prvky komponenty `EventListElement`. EventListElement je zase také samostatná součástka, která obsahuje funkce určené pro práci s prvkem „Událost“.

EventListElement

EventListElement je komponenta, která je zodpovědná za všechny základní operace s událostí. Ona ošetřuje i interakce uživatele s položkou seznamu. Je primárně odpovědná za mazání a aktualizaci záznamů. Kromě toho je odpovědná za‘reakci na akce uživatelů, jako je například kliknutí. Po kliknutí na událost se zobrazí vyskakovací okno s podrobnými informacemi o této události.

Přidání, aktualizace a odstranění události

V této části se dotkneme diskuze o třech klíčových funkcích kalendáře, bez kterých si kalendář nelze představit.

Přidání události

V měsíčním kalendáři tuto akce lze provést několika způsoby a zde se podrobně podíváme na každý z nich.

První a nejpopulárnější způsob přidání nových událostí do kalendáře je prostřednictvím formuláře.

Formulář pro přidání kalendáře lze otevřít kliknutím na tlačítko v pravé dolní části stránky. Kliknutím na toto tlačítko se zobrazí vyskakovací okno obsahující formulář. Tato komponenta byla vyrobena pomocí knihovny React Material UI.

Formulář obsahuje pouze tři pole. První pole je pro zadání názvu události, další dvě pro datum před a po. Všechna pole obsahují požadované ověření. Název formuláře se mění v závislosti na tom, zda chceme vytvořit novou událost nebo aktualizovat existující. Vyplňování formuláře se provádí pomocí `useState`. Po odeslání formuláře jsou data serializovaná do formátu JSON a odeslána na server v jedné ze zpráv webosocketu.

Druhá metoda je uživatelům také známá - jednoduše můžete myší natáhnout požadovaný rozsah dat a v tuto chvíli se také otevře formulář pro přidání události. Tento formulář již bude obsahovat uživatelem vybrané časové období.

Aktualizace událostí

.

Aktualizace událostí je ve skutečnosti velmi podobná vytvoření nové, ale zde je z důvodu pohodlí uživatele nutné do formuláře přenést data, která již dříve zadal. K tomu potřebujeme přijímat data ze serveru. Zde klient odešle serveroví zprávu o načítání dat, která byla zmíněna dříve. Po načtení stránky klient odešle stejnou zprávu na server. Jako odpověď klient obdrží JSON zprávu, která obsahuje aktualizovaná data. Tato zpráva se předá zpět funkci, která zobrazuje události, a na obrazovce se zobrazí seznam těchto událostí.

Formulář pro přidání / aktualizaci události v měsíčním kalendáři obsahuje pole pro datum pouze ve formátu samotného data, tj. bez pole pro hodiny. Funkce na serveru, která

přijímá data z formuláře, je však jednotná, proto se chybějící část, která obsahuje hodiny, minuty a sekundy, generuje samostatně, aby se požadavek přenesl do formuláře jednotným.

Mazání událostí

Nejjednodušší z přítomných funkcí. Tato funkce odesílá ID prvku, který má být odebrán, na šerver a ve výsledku přijímá redukované datové pole. Tato data je nyní třeba znovu zobrazit na obrazovce, aby uživatel viděl, že událost byla skutečně odstraněna.

Kontextová nabídka

Po kliknutí pravým tlačítkem na událost se uživateli otevře kontextové menu. Tato nabídka byla také vytvořena pomocí knihovny uživatelského rozhraní React Material UI a obsahuje 2 tlačítka - aktualizaci a odstranění události. Kontextová nabídka není přetížena zbytečnými funkcemi, aby co nejvíce urychlila práci uživatele s aplikací. Za každým z těchto tlačítek na klientovi je funkce, která zpracovává a připravuje data pro odeslání na server. Po kliknutí na jednu z položek nabídky bude požadavek odeslán a stránka bude překreslena.

Výše byla popsána implementace samotného kalendáře pro měsíc a jeho hlavní funkce. Dále bude popsána hlavní část kalendáře, konkrétně kalendář na týden.

Týdenní kalendář

Týdenní kalendář má klasický vzhled a je podobný ostatním kalendářům. Toto je asi nejpohodlnější formát. Celý kalendář na týden je vykreslen v komponentách umístěných v adresáři týdne (Week). Kořenová složka se zde jmenuje `WeekView`. Tento vzhled má následující klíčové komponenty:

- Blok s hlavičkou
- Samotné tělo kalendáře

Blok s hlavičkou

Tento blok obsahuje tři prvky. Název měsíce a dvě tlačítka pro přepínání týdne. Samotná data se zobrazují pomocí knihovny `moment` jazyka JavaScript. Tato knihovna umožňuje manipulovat s daty na jakékoli úrovni včetně úrovně týdnů. Po načtení stránky se týden otevře v aktuálním dne.

Blok obsahující týdenní kalendář

Tento blok je představen ve formě tabulky, kde sloupce jsou dny v týdnu a řádky jsou hodiny. Každý ze sedmi dnů týdnu má 24 hodiny.

Když se stránka načte, klient také odešle požadavek na server, aby získal seznam událostí z aktuálního kalendáře. Pokud seznam není prázdný, události se zobrazí v příslušných buňkách.

Kreslení týdenního kalendáře

Týdenní kalendář se zobrazuje v komponentě Týden. Každý den v týdnu je zase reprezentován nezávislou složkou `WeekDay`. Zaměřme se na jeho popis, protože o něj je největší

zájem. Komponenta `WeekDay` se skládá ze dvou částí, z nichž každá slouží k zobrazení událostí. První je takzvaný název. Obsahuje název dne v'týdnu a hlavně datum. Toto datum nastaví jedinečné ID pro celou komponentu a bude to datum, který bude později použitý u potomků.

Druhá část komponenty WeekDay v zásadě obsahuje seznam 24 položek, které tvoří 24 hodiny dne. Tyto komponenty mají jedinečné ID, ale toto ID je jedinečné pouze v rámci komponenty `WeekDay`. Toto ID později bude stanovit denní dobu pro kreslení událostí. Pokud potřebujeme odkazovat na buňku v rámci kalendáře, bude její ID vygenerován samostatně na základě ID celého dne.

Zobrazování událostí

Jak je popsáno výše, události přicházejí ze serveru v odpovídající websocket zprávě. Tato zpráva je přijímána jako vstup centrální funkcí `renderWeek`. Tato funkce, stejně jako ostatní vykreslovací funkce, převádí výsledné pole do vlastního formátu, který lze poté pohodlně namapovat na odpovídající hodiny nebo i na případně komponenty `EventElement`. Dbá hlavně na to, aby byla data v konzistentním formátu. Komponenta `EventElement` je obdobou komponenty `EventListElement` z měsíčního kalendáře. Ale kvůli některým významným rozdílům ve formátu zobrazení a způsobu provádění funkcí, byl odebrán jako samostatná součástka. I když ve skutečnosti provádí stejné operace jako v měsíčním kalendáři, avšak trochu jiným způsobem.

Přidání události Stejně jako v měsíčním kalendáři existuje několik způsobů, jak přidat událost. První z nich, stejně jako na předchozí stránce, umožňuje přidat událost pomocí formuláře, tlačítko, které jeho otevírá, je také umístěno v pravé dolní části stránky. Tentokrát však již nepotřebujeme pole pro zadání data, ale pouze pole pro zadání času. Uživatel musí být stále schopen vybrat datum, takže ve formuláři zůstalo jedno pole. Po odeslání formuláře se data sjednotí a odešlou na server pak klient po obdržení odpovědi překreslí týdenní kalendář.

Druhým způsobem vytváření nových událostí je opět to, že uživatel si jednoduše natáhne hodiny, ve kterých chce vytvořit událost, a otevře se formulář pro přidání události, ve kterém bude mít nyní vybraná data. Tato data budou také přeložena do požadovaného formátu.

Drag and drop

Týdenní kalendář je implementován jako *Drag and Drop* součástka. Jakmile jsou události vytvořeny, lze je snadno přetahovat mezi jednotlivými hodinami a dny v týdnu. Při takovém pohybu událostí budou informace o nich přeneseny na server jako aktualizační zpráva. Události přesahující interval jednoho týdne však bude nutné vytvořit v měsíční verzi kalendáře.

Aktualizace a mazání událostí

Aktualizace a mazání se provádí stejným způsobem jako v měsíčním kalendáři. Jediným rozdíl je v tom, že zde opět klient pracuje s mírně odlišnými daty a ke zpracování těchto je mu zapotřebí dalších funkcí.

4.3 Databáze

Databáze obsahuje pouze dvě tabulky. Tabulka `Events` a tabulka `Calendars`. Databáze zpočátku obsahovala také tabulku `Users`, ale tato tabulka neodpovídala konceptu aplikace,

aplikace by neměla ukládat kalendáře. Uživatel pokaždé pracuje s samostatným kalendářem a neměl by mít přístup k seznamu svých kalendářů.

4.4 Testování

Kvůli jednoduchosti serverové strany aplikace a skutečnosti, že většina prací na projektu probíhala na klientovi, nebyl backend pokrytý testy. Některá omezení vstupu byla přidána ručně a kód se během vývoje nezměnil. Z tohoto důvodu nebylo nutné jeho často kontrolovat.

Klientská strana byla ručně testována prvními uživateli. Během celého procesu vývoje byly nově implementované části testovány na použitelnost, vzhled a úplnost funkčnosti. Podobně testování mělo určitý dopad na vývojový tok.

4.5 Nasazení

Aplikace byla zveřejněna na hostingu Heroku. To je populární hosting, který vám umožní snadno umístit vaše aplikace na internetu. Heroku má jednoduché a intuitivní rozhraní a také rozsáhlou dokumentaci, ze které se uživatel může hodně naučit. S Heroku můžete pracovat z příkazového řádku. K tomu oni mají aplikaci Heroku CLI. Ve skutečnosti se jedná o klasické úložiště Git, které podporuje všechny příkazy Gitu a několik dalších Heroku přikážu navíc.

Abyste mohli na Heroku umístit soubory, musíte se nejprve zaregistrovat a vytvořit si projekt. Při vytváření projektu můžete vybrat jeho typ a také typ databáze, která bude v projektu použita. Poté, co projdete procesem vytváření aplikace, bude úložiště Git uživateli k dispozici. Budete muset nahrát soubory aplikace do tohoto adresáře. To však zatím nebude stačit. Zvláštností aplikací na Heroku je, že pro svou práci vyžadují tzv. *Procfile*. Tento soubor nemá žádnou příponu a je vždy umístěn v kořenovém adresáři aplikace.

Tento soubor popisuje, jak bude aplikace spuštěna. Ke spuštění aplikace jsem použila server *gunicorn*. Je to WSGI server Pythonu pro Unix. Právě na tomto operačním systému běží Heroku. Gunicorn je založen na *pre-fork* modelu. To znamená, že máme jeden centrální proces, který spravuje tzv. *worker processes*. Master o klientech ale nic neví. Workers mohou být synchronní nebo asynchronní. Aplikace s websockety vyžaduje pouze asynchronního workera.

Aby se aplikace mohla spustit, musíme do kořenového adresáře umístit soubor s požadavky - *requirements.txt*. Poté, když nahrávaní aplikace skonči, bude Heroku odkazovat na tento soubor za účelem instalace všech balíčků, které budou v aplikaci použity. Takový soubor se generuje automaticky a například v Pythonu jej lze vygenerovat příkazem `pip freeze requirements.txt`.

Jakmile je vše připraveno, když Heroku dokončí instalaci knihoven a balíčků ze souboru `requirements.txt`, aplikace je připravena k použití. Zůstane jen přenavigovat na dříve vybranou adresu a po chvíli se aplikace spustí. Při prvním spuštění aplikace trvá trochu dlouho. Je to způsobeno skutečností, že když aplikace byla zapnuta poprvé nebo pokud od posledního spuštění uplynulo více než 30 minut, server se vypne a musí být spuštěn znovu.

Heroku má dobrou podporu pro databáze PostgreSQL. Přímo na tomto hostingu můžeme spustit SQL server, který bude aplikaci obsluhovat. To lze provést také několika kliknutími a výběrem verze databáze. Poté musíte do kódu aplikace přidat *connection string*.

Kapitola 5

Výsledky

Výsledkem této bakalářské práce bylo vytvořeni webové aplikace kalendáře, která umožňuje extrémně snadné sdíleni. Samotný vývoj sestával z několika fází:

1. Tvorba databázového schématu a tabulek.
2. Generování kódu serveru
3. Vytvoření klientské strany aplikace pro samostatné vzhledy.

Při programování této aplikace jsem pro sebe udělala následující závěry:

- Ujistila jsem se, že jazyk Python, i když se dnes používá hlavně pro umělou inteligenci a matematické výrazy, je vhodný i pro programování počítačových webových aplikací.
- Nejlepším řešením je napsat frontend aplikace pomocí webového frameworku. Ten organizuje a strukturuje váš kód, přičemž ponechává prostor pro budoucí rozšíření a doplňky vaší aplikace.
- Při práci na této aplikaci jsem se také naučila plánovat velké osobní projekty.

Aplikace podle mě splňuje požadavky stanovené na začátku práce. Ale přestože je základní funkce v této aplikaci implementována, chtěla bych se v této části zaměřit na body, které lze stále dokončit nebo vylepšit. Nejprve bych chtěla pokračovat v práci na optimalizaci aplikace. Nyní bych přepsala některé funkce jinak, protože jejich logika se mi nyní zdá příliš komplikovaná a některé části nejsou vůbec potřeba.

Druhá věc, na které bych chtěla pracovat, je změna stylů kalendáře. Pokud budu i nadále vyvíjet tuto aplikaci, pak by stálo za to dotáhnout její styly k dokonalosti, aby byl kalendář pro uživatele opravdu atraktivní. Dále bylo by přidat některé funkce, jako je změna barev událostí, protože tím bude kalendář všestrannější. Zbytek bych nechtěl přetížit aplikaci, podle mého názoru by měla zůstat stejně minimalistická.

Považuji ovsem za nevhodné přidávaní uživatele do kalendáře, protože by to narušilo nezávislost kalendářů na sobě - uživatel bude mít přístup k seznamu svých kalendářů nebo k historii úprav. Ale tento kalendář byl vytvořen tak, aby jej skupina lidí mohla snadno a rychle upravovat bez nutnosti registrace a všichni uživatelé by měli mít stejná přístupová práva prostřednictvím pouze odkazu. Během procesu vývoje byl také výrazně snížen objem kódu serveru. Z původně velkého počtu funkcí nakonec zůstalo jen několik. Ale tato malá částka byla optimalizována, aby nyní mohli na stránce provádět jakékoli akce, které potřebuji.

To je pro daný okamžik vše, co bych chtěla změnit nebo přidat. Nechci, aby aplikace byla příliš velká a nechtěla bych, aby se nekonečně vylepšovala. Vždy je co dodat, ale ráda bych zachovala původní myšlenku tohoto kalendáře, aby tam zůstalo jen minimum nezbytných funkcí.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit webový kalendář umožňující extrémně snadné sdílení. V této práci najde čtenář popis technologií, které byly použity k vytvoření zmíněné aplikace. Pro každou z hlavních technologií nebo knihoven byl napsán krátký přehled. Také v této práci je popsán samotný princip realizace zadaných úkolu. Díky tomuto popisu bude čtenář schopen dozvědět se o důvodech výběru konkrétní implementační metody. Aplikace byla vytvořena pomocí JavaScriptu a jeho knihovně React na straně klienta. Server byl plně implementován v Pythonu a jeho mikroframeworku Flask. Jako databáze byla vybrána relační databáze PostgreSQL. Samotná aplikace byla zveřejněna na hostingu Heroku a je nyní k dispozici na adrese https://quickycal.herokuapp.com/. Dokumentace a příklady kódu jsou k dispozici na GitHub.

Tato aplikace implementuje některá svá vlastní řešení pro úkoly vykreslování kalendáře a správy jeho událostí, což může být užitečné při vylepšování a doplňování této nebo nějakých jiných aplikací. Tyto řešení byly popsány v kapitole která se věnuje implementaci.

Při vykonaní této bakalářské práce jsem se naučila hlavně vytvářet aplikaci v Reactu, což mi otevřelo mnoho příležitostí k tvorbě moderních webových aplikací. Při této práci jsem opravdu prohloubila své znalosti o technologii, která pro mě byla dříve skoro nová. Také jsem nahlédla dovnitř principu webových kalendářů a vystudovala jsem moderní způsoby jejich implementace. Vytvoření této aplikace bylo velmi zajímavé a zábavné.

V této práci se mi podařilo celkem úkol zrealizovat. Samozřejmě ji lze stále vylepšovat a výběr technologií zohledňoval následná vylepšení a rozšíření. Avšak již nyní je to plnohodnotná aplikace, která může vykonávat své funkce a uživatelé ji mohou používat. Aplikaci v budoucnu lze doplnit o širší škálu funkcí a případně ji rozšířit, ale nerada bych ji přetěžovala, aby se nepodobala stávajícím aplikacím a jejichž nevýhody způsobily vznik tohoto projektu. Možná, že některé funkce nebyly implementovány ideálně a teď bych některé z nich psala odlišně, změnila bych například trochu logiku zpracování událostí, ale obecně si myslím, že výběr technologií je správný a umožňuje nám provádět změny v projektu bez jakýchkoli problémů. Kromě toho bych určitě pracovala bych dále na stylech, nebo by je mohl upravovat profesionál, protože hlavní věcí v takových aplikacích je jejich vizuální reprezentace.

Literatura

[1] ANDREWSONIN. *Hlavní nevýhody jazyka Python* [online]. 2020 [cit. 2021-05-05]. Dostupné z: https://habr.com/en/post/504126/.

[2] CANVA. *The history of web design* [online]. 2021 [cit. 2021-05-05]. Dostupné z: https://www.canva.com/learn/web-design-history/.

[3] FREEMAN, J. *What is JSON? A better format for data exchange* [online]. 2019 [cit. 2021-05-05]. Dostupné z: https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html.

[4] GACKENHEIMER, C. *What Is React?* Apress, 2015. ISBN 978-1-4842-1246-2.

[5] INC., F. *Introducing JSX* [online]. 2021 [cit. 2021-05-05]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

[6] INVERITA. *The Best Web Application Development Languages in 2021* [online]. 2021 [cit. 2021-05-05]. Dostupné z: https://medium.com/@inverita/the-best-web-application-development-languages-in-2021-6b6eb5944925.

[7] KLEIN, B. *Packages in Python* [online]. 2014 [cit. 2021-05-05]. Dostupné z: https://www.python-course.eu/python3_packages.php.

[8] M., G. *Flask-SocketIO* [online]. 2014 [cit. 2021-04-24]. Dostupné z: https://flask-socketio.readthedocs.io/en/latest/.

[9] NET INFORMATIONS. *The Pros and Cons of JavaScript* [online]. 2019 [cit. 2021-05-05]. Dostupné z: http://net-informations.com/js/iq/advan.html.

[10] RONACHER, A. *Foreword* [online]. 2013 [cit. 2021-04-24]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/foreword/.

[11] WANG, V., SALIM, F. a MOSKOVITS, P. *The Definitive Guide to HTML5 WebSocket*. Apress, 2013. ISBN 143024741X, 9781430247418.

Příloha A

Obsah datového media

Přiloženy datový medium obsahuje následující soubory:

- zdrojové kódy
- knihovny potřebné pro překlad,
- přeložené řešení,
- PDF s technickou zprávou,
- krátká dokumentace
- zdrojový kód technické zprávy.